DOULOS

Delivering KnowHow www.doulos.com

Designing to the Worst-Case Scenario

Practical System Call Filtering with Seccomp

Presenter: Simon Goda

Senior Member Technical Staff

Doulos

Introduction

- About me:
  - Background in applied maths, algorithm development, 3D graphics, …
  - Worked for STMicroelectronics in Bristol, UK for nearly 15 years
    - Supporting and training customers using Linux and RTOS on set-top box and home entertainment products
  - Joined Doulos in 2014, write and deliver training in the embedded Linux space, including device drivers, Yocto and Linux security

- About Doulos:
  - Providing training in hardware design, verification, embedded software & deep learning for over 30 years
  - Offer scheduled classes and bespoke team training both in-person and live online
  - Employee owned since 2022

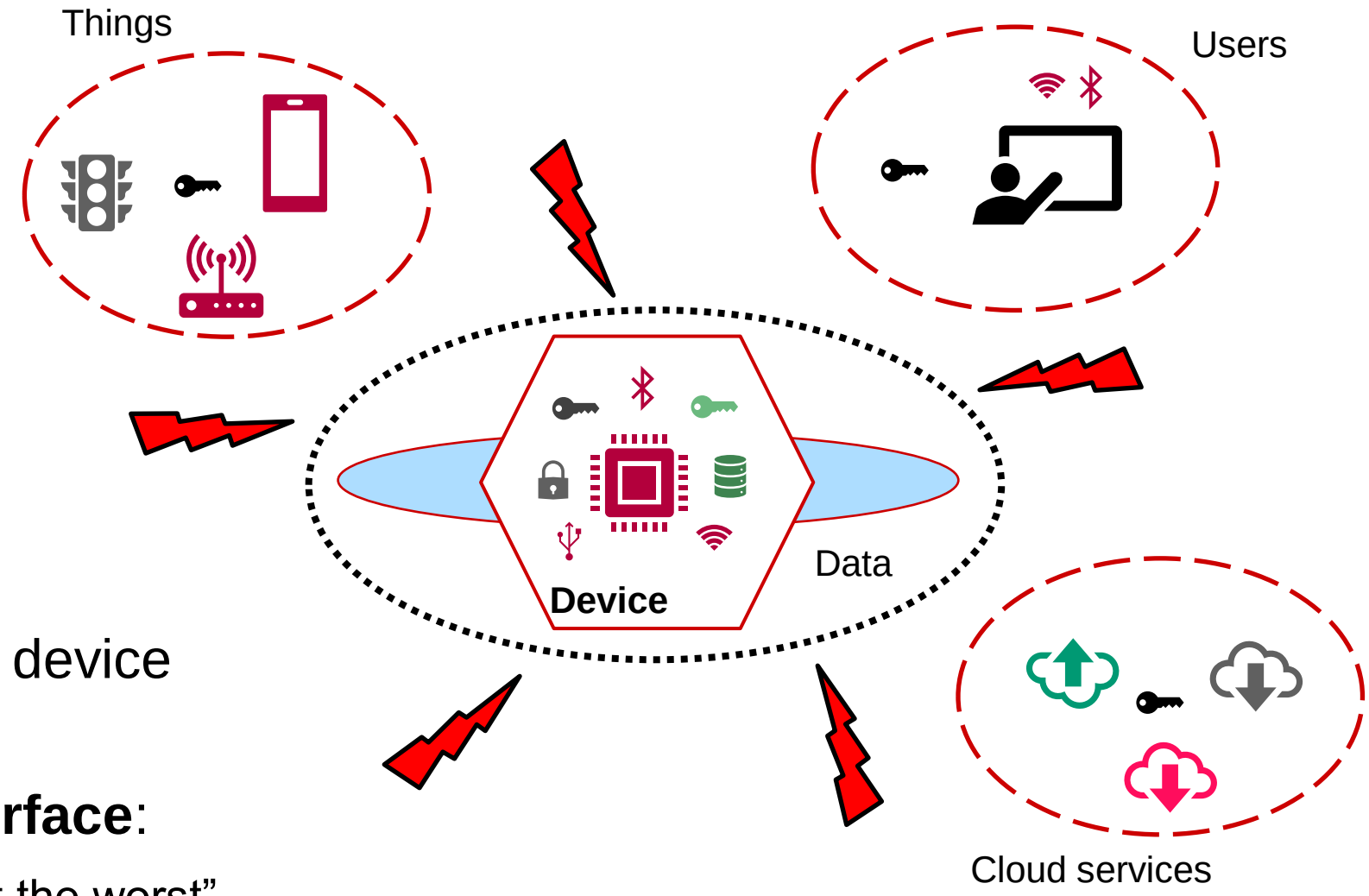
Practical System Call Filtering with Seccomp

- Embedded Linux Systems & Security
- Basic Concepts
- Syscall Filtering with Libseccomp:
  - Configuring the filters
  - Which Syscalls?
- Seccomp in Practice:
  - systemd
  - LXC
  - Docker

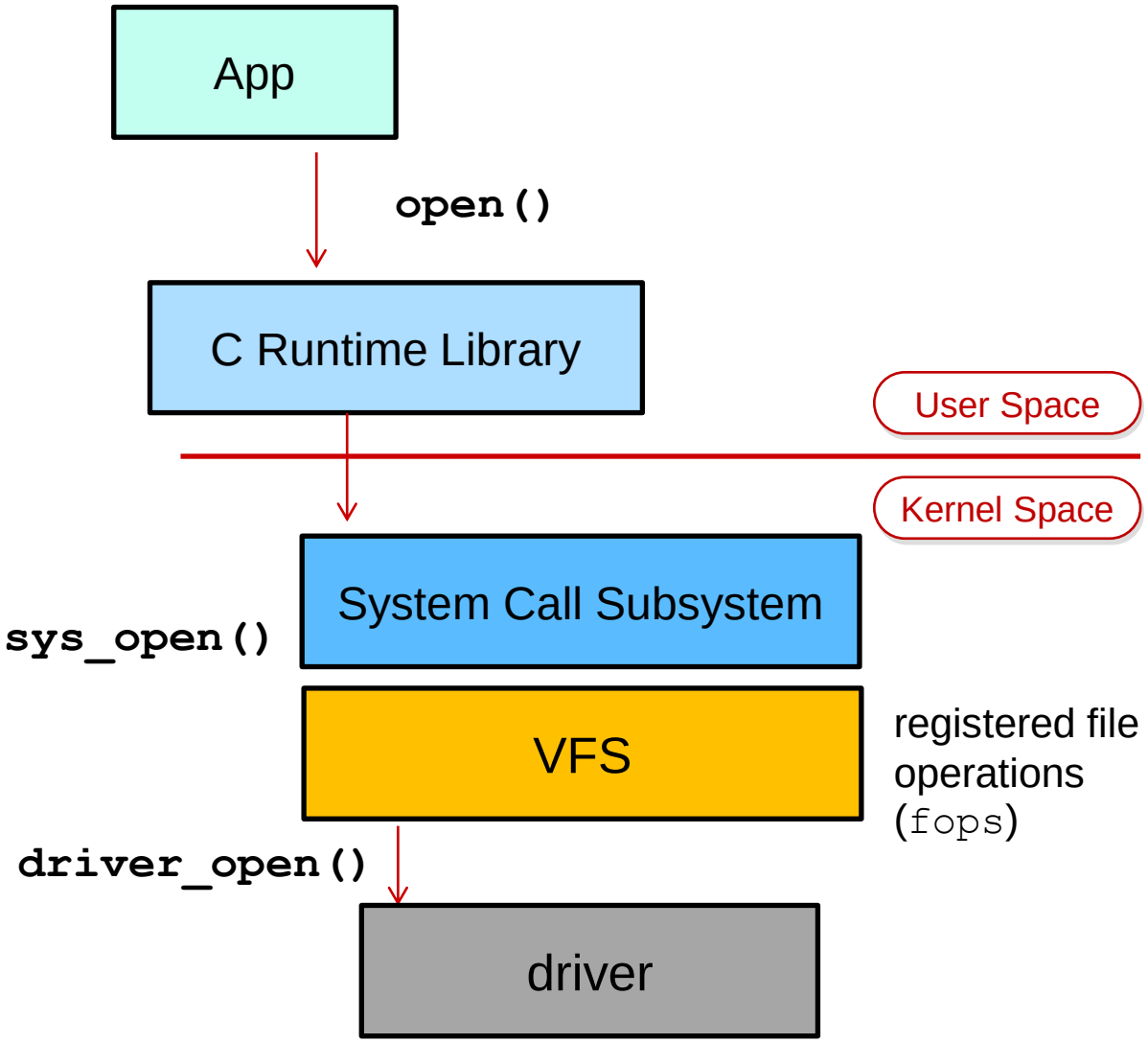
DOULOS

SECURE EMBEDDED

- Intended use
  - Requires correctness
- Misuse possible if
  - Flawed design
  - Software weaknesses
- Security protects against device misuse
- Minimizing the **attack surface**:
  - “Hope for the best, expect the worst”



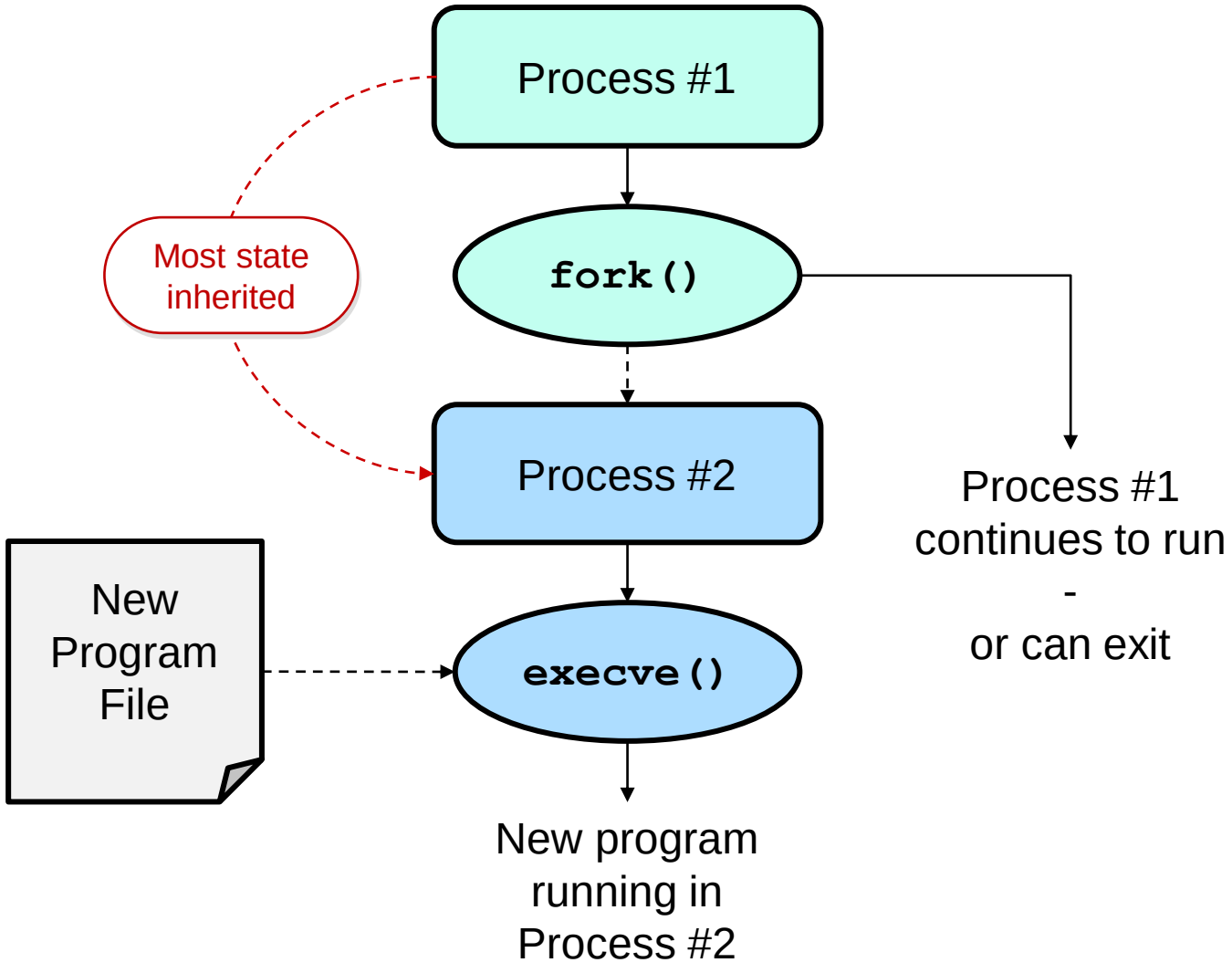
User Space ↔ Kernel Space: The System Call Interface



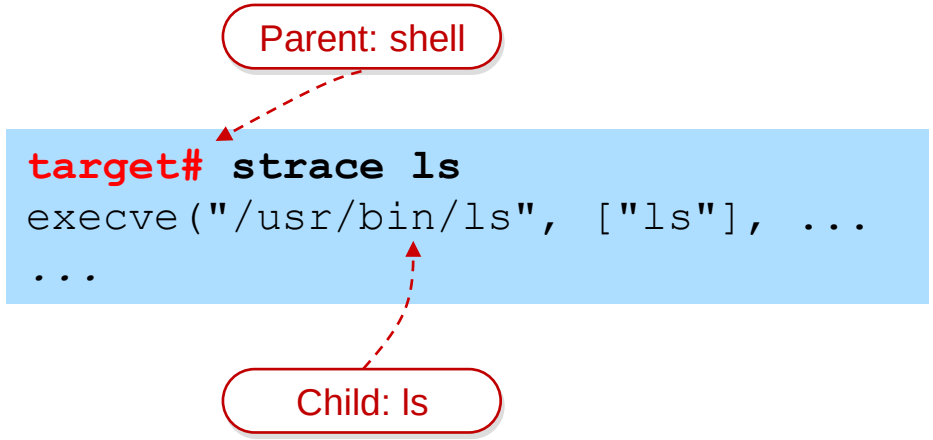
- Applications in *user space* make requests to the *kernel* via **system calls**
  - Usually via corresponding library call
- **~300** system calls available

- User space **can't** interact with hardware or resources directly
  - Access is controlled by the kernel (highest privilege)
  - User space processes at lowest privilege level

Parent and Child Processes

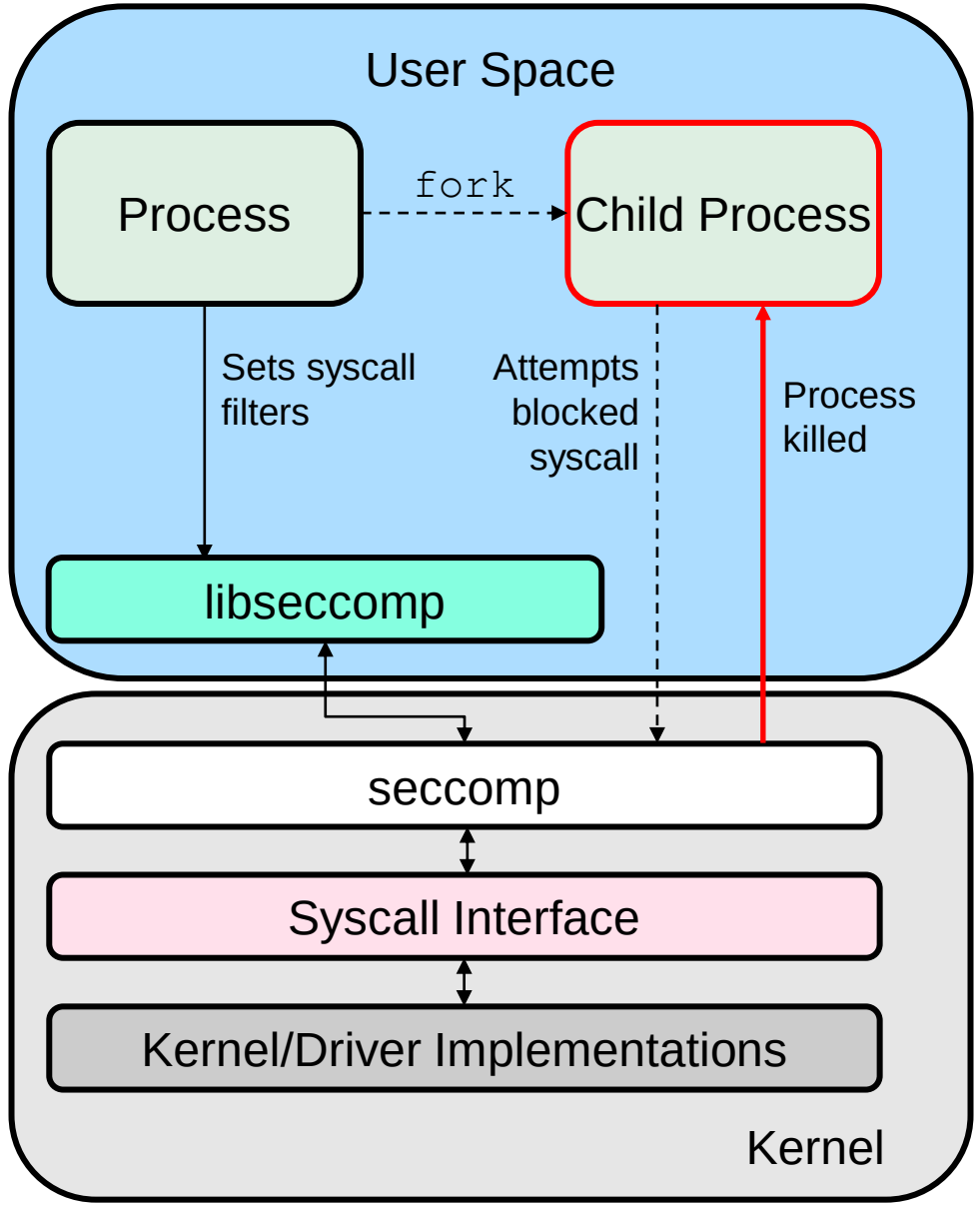


- New user space processes are created by a *parent*
  - e.g. the **shell** or the **init** process
- New *child* process then loads the required program



Syscall Filtering with `libseccomp`

- Kernel can filter syscalls to block unauthorised calls
  - seccomp uses BPF code
  - Enabled with `CONFIG_SECCOMP`
- **libseccomp** provides user space APIs
- Filters set per process
- Inherited by child processes



More on libseccomp

- Actions:
  - SCMP_ACT_KILL
  - SCMP_ACT_KILL_PROCESS
  - SCMP_ACT_TRAP
  - SCMP_ACT_ERRNO
  - SCMP_ACT_TRACE
  - SCMP_ACT_LOG
  - SCMP_ACT_ALLOW
  - SCMP_ACT_NOTIFY

- Main APIs:

```
#include <seccomp.h>

/* initialize filter struct */
scmp_filter_ctx ctx = NULL;

/* set default action */
ctx = seccomp_init(SCMP_ACT_KILL);

/* add exceptions */
rc = seccomp_rule_add(...);

/* load filter into kernel and start */
rc = seccomp_load(ctx);
```

Configuring the Filters

- Very simple example:

Set in the controlling 'parent' application

```
#include <seccomp.h>

scmp_filter_ctx ctx = NULL;

ctx = seccomp_init(SCMP_ACT_KILL);

rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0);

rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);

rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(close), 0);

rc = seccomp_load(ctx);
```

set action for unauthorised syscalls

allow read, write and close only

converts syscall name to arch specific number

load filter into kernel

Configuring the Filters

- Also possible to filter based on syscall parameter values:

```
int fd;
unsigned char buf[BUF_SIZE];
...
fd = open("file.txt", O_RDONLY);
...
rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 3,
                      SCMP_A0(SCMP_CMP_EQ, fd),
                      SCMP_A1(SCMP_CMP_EQ, (scmp_datum_t)buf),
                      SCMP_A2(SCMP_CMP_LE, BUF_SIZE));
```

number of arguments to check

all of the comparisons must be true

SCMP_A0 - argument 0 etc.

SCMP_CMP_NE
SCMP_CMP_LT
SCMP_CMP_LE
SCMP_CMP_EQ
SCMP_CMP_GE
SCMP_CMP_GT
SCMP_CMP_MASKED_EQ

Which Syscalls?

- Use seccomp logging to list active system calls:
  - Set in controlling 'parent' application:

```
/*ctx = seccomp_init(SCMP_ACT_KILL);*/
ctx = seccomp_init(SCMP_ACT_LOG);
```

  - Then run the processes and check the logs:

```
target# journalctl | grep SECCOMP
...
Jul 21 12:51:37 qemuarm64 audit[261]: SECCOMP auid=4294967295 uid=0 gid=0 ses=4294967295
subj=_ pid=261 comm="hello" exe="/home/root/hello" sig=0 arch=c00000b7 syscall=57
compat=0 ip=0x7f88ac84fc code=0x7ffc0000
...
```

- Then use `scmp_sys_resolver` to find syscall names:
  - Part of libseccomp package

```
target# scmp_sys_resolver -a aarch64 -t 57
close
```

Which Syscalls?

- **strace** can also be used to generate lists of system calls:

```
target# strace <app>
...
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@p\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=163200, ...}) = 0
...
```

  - `strace -ff` will trace child processes and write results to separate files
    - Can generate syscalls lists for containers on launch
- Some command line magic* to generate a simple list:

```
strace -c -f -S name <app> 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'
...
access
arch_prctl
brk
close
...
```

* From: https://github.com/docker/labs/blob/master/security/seccomp/README.md

Simple Example

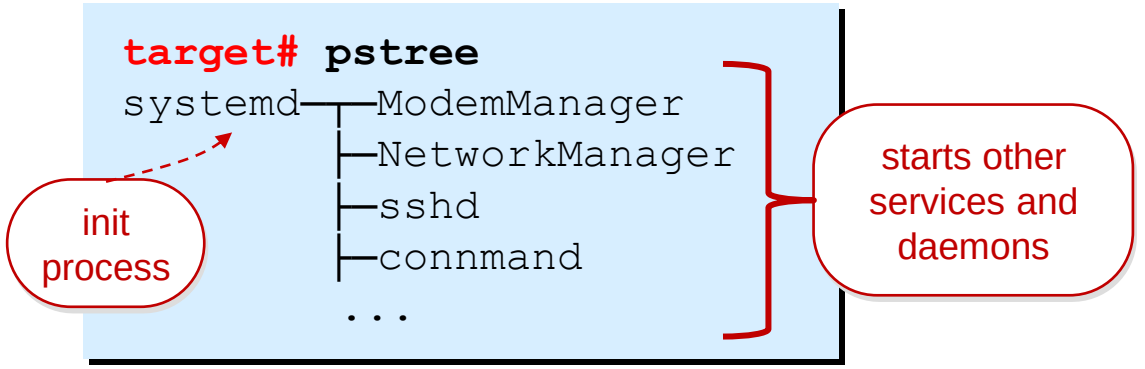
```
/* Controlling Parent Application */

#include <seccomp.h> /* Headers */
...

int main(void) {
    ...
    /* set up seccomp filtering */
    ctx = seccomp_init(SCMP_ACT_KILL);
    ...
    rt = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(clone), 0);
    ...

    /* create child process */
    childPid = fork();
    execl("/home/root/hello", "/home/root/hello", ".", (char *) NULL);
    ...

    wait(&status); /* Parent waits for child process to terminate */
    ...
}
```

```
/* Child Application */

/* No special Headers Needed */

int main(void) {
    /* Just prints PID */
    /* and exits */
    ...
}
```

compile with `-lseccomp`

Seccomp in Practice: systemd

- The **init** process is the first user space process
  - Has PID = 1
  - Starts other user space processes
- Most modern systems use **systemd**



- **systemd** can set syscall filters per service
  - Either by individual system call
  - Or using predefined system call sets

```
target# cat /etc/systemd/system/test.service
[Unit]
Description=Test Service
After=network.target

[Service]
User=root
Group=root
ExecStart=/usr/bin/testdaemon
SystemCallFilter=~uname

[Install]
WantedBy=multi-user.target
```

block calls to uname

[● ◀] systemd

Seccomp in Practice: systemd

- Predefined sets include:

`@basic-io`	`@chown`	`@clock`	`@debug`	`@file-system`	`@io-event`
`@module`	`@mount`	`@network-io`	`@process`	`@setuid`	`@system-service`

- Best to use ‘allow’ lists e.g.

```
SystemCallFilter=@file-system @basic-io
```

Filesystem and basic IO operations allowed

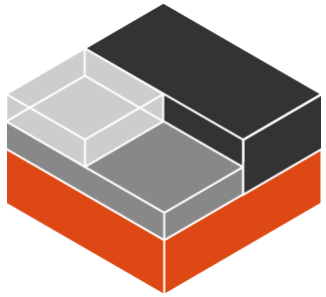
  - Basic functionality (`execve`, `exit`, etc.; sleeping; time querying) allowed by default

- ‘deny’ lists can be created with **~** which inverts the filter (i.e. blocks)

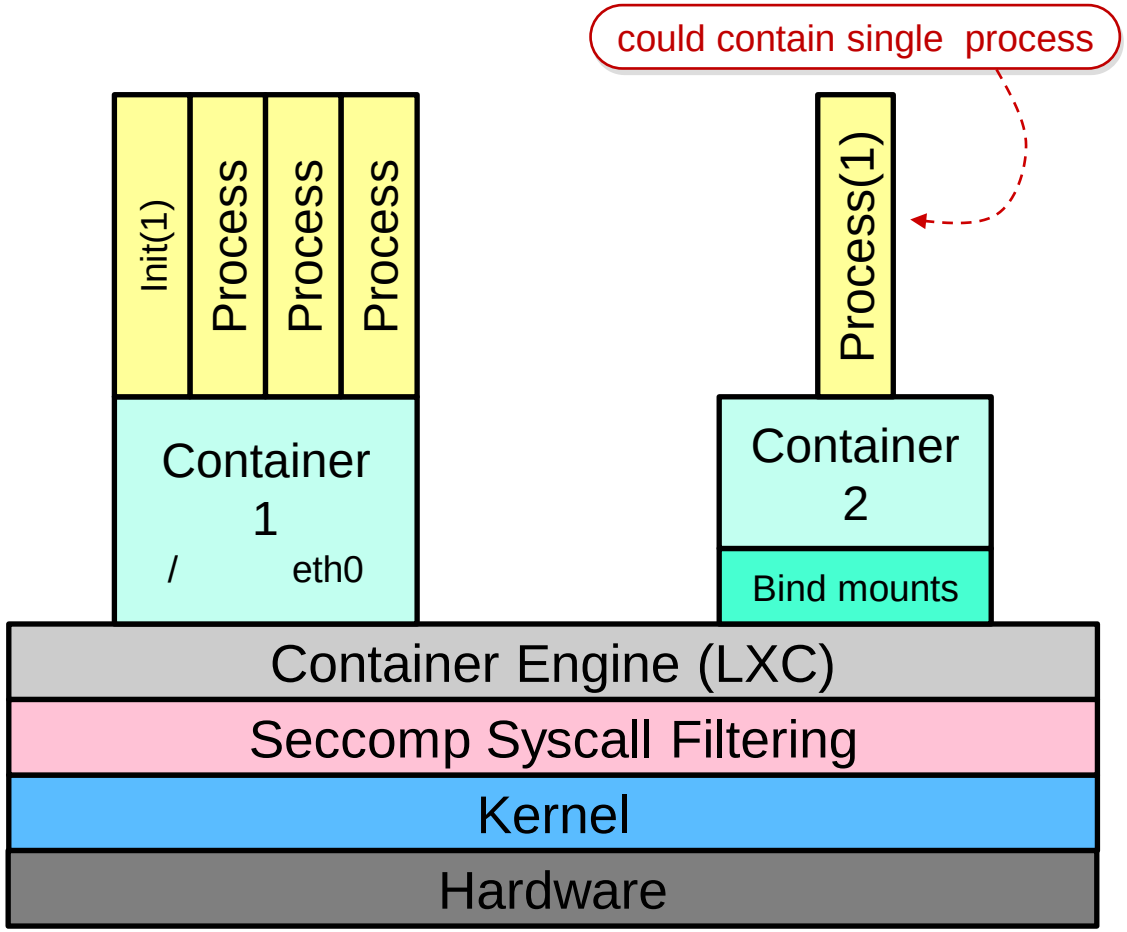
```
SystemCallFilter=~@debug ~@clock
```

Seccomp in Practice: Containers

- Isolated execution environments such as **LXC** & **Docker** can configure syscall filtering
- ‘Lightweight’ or single process containers can be ‘locked down’
- More functional containers can be blocked from making ‘unusual’ syscalls
  - Reduce the ‘attack surface’

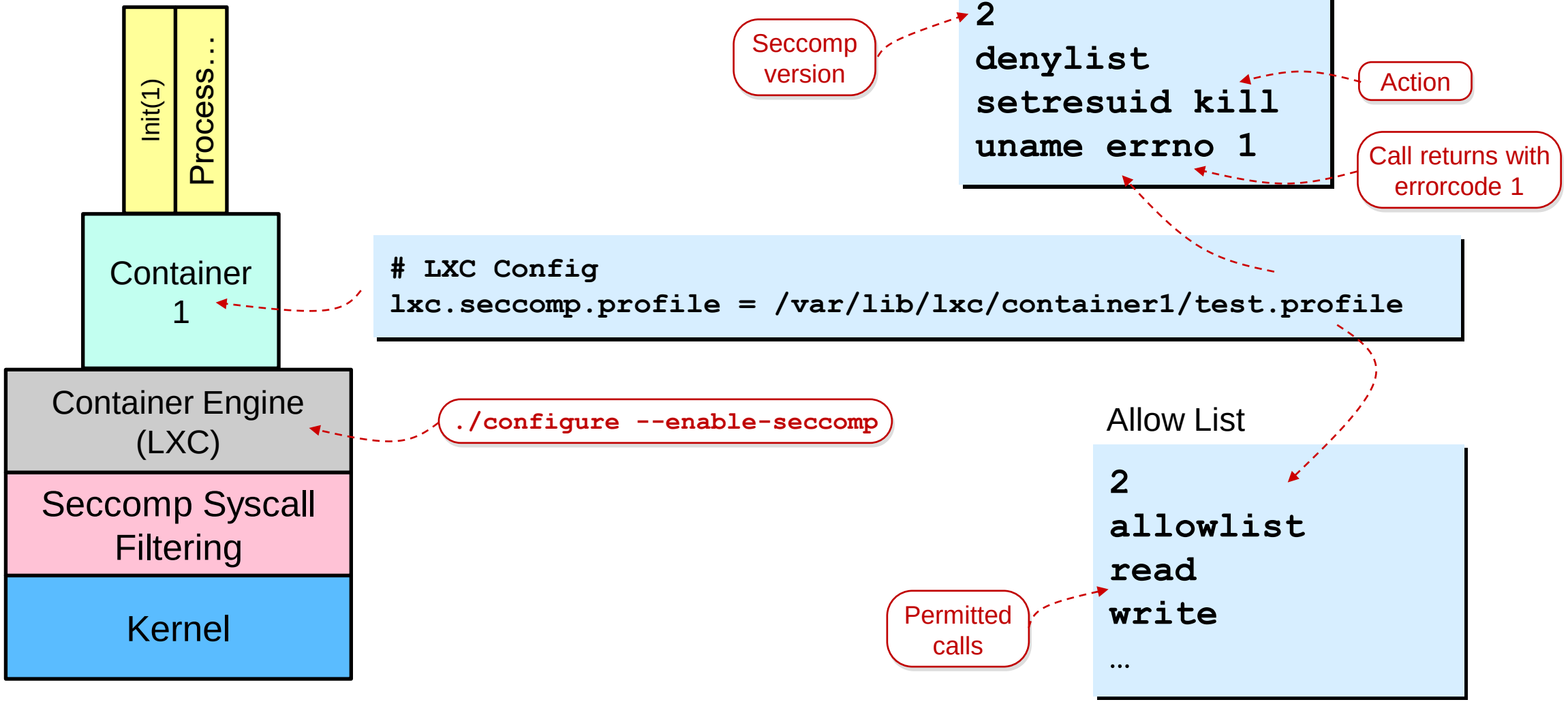


docker



Seccomp in Practice: LXC

- To use Seccomp with LXC containers:



Seccomp in Practice: Docker

- Docker automatically runs containers with a default Seccomp filter
  - Blocks around 44 risky and/or obsolete syscalls, e.g.
    - `bpf`, `clone`, `kexec_load`, `mount`, `settimeofday`, `unshare`, ...
- Blocked calls return `Permission Denied`
  - i.e. default action is `SCMP_ACT_ERRNO`
- Can run without the default profile:

```
target# docker run --rm -it -security-opt seccomp=unconfined ...
```

  - Not recommended!

Seccomp in Practice: Docker

- Can define custom profile:

```
target# docker run --rm -it -security-opt seccomp=custom.json ...
```

```
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "architectures": [
        "SCMP_ARCH_AARCH64"
    ],
    "syscalls": [
        {
            "name": "uname",
            "action": "SCMP_ACT_KILL",
            "args": []
        },
        ...
    ]
}
```

```
SCMP_ARCH_X86_64
SCMP_ARCH_RISCV64
...
```

```
SCMP_ACT_ALLOW
SCMP_ACT_KILL
SCMP_ACT_TRAP
SCMP_ACT_ERRNO
SCMP_ACT_TRACE
```

Can filter by syscall arguments, e.g.

```
"args": [
    {
        "index": 0,
        "op": "SCMP_CMP_MASKED_EQ",
        "value": 2080505856,
        "valueTwo": 0
    }
]
```

Confining Linux Applications with Libseccomp

- Seccomp/Libseccomp provides a powerful mechanism for limiting the functionality of a Linux application or process
- The system is protected from hacked or modified programs
- Syscall filtering can be managed:
  - per-process
  - by the systemd init program
  - in isolated executions environments using LXC or Docker

- Seccomp is one small part of securing a Linux system....

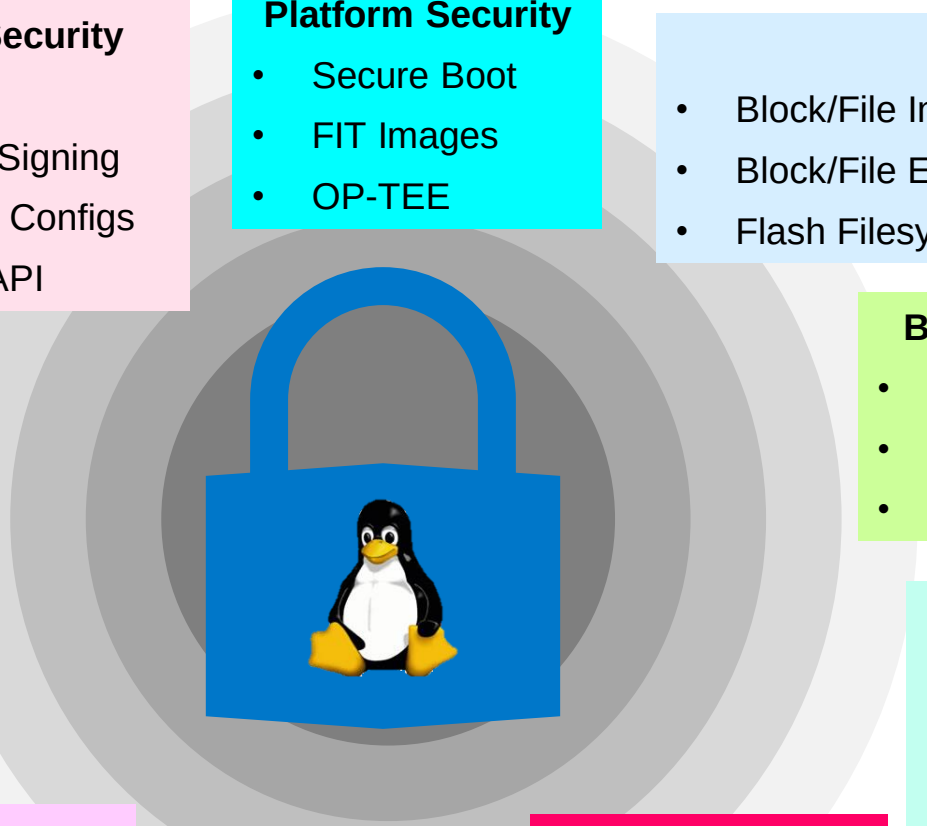
Seccomp: One Piece of the Linux Security Jigsaw

OS Features
- DAC (permissions)
- MAC (SELinux etc.)
- Capabilities
- Systemd
- **Seccomp**

Kernel Security
- KASLR
- Module Signing
- Security Configs
- Crypto API

Platform Security
- Secure Boot
- FIT Images
- OP-TEE

Filesystem Security
- Block/File Integrity (dm-integrity, dm-verity, fs-verity)
- Block/File Encryption (dm-crypt, fscrypt)
- Flash Filesystems (UBIFS)



Build System Tools
- Yocto
- meta-security
- meta-selinux

Languages & Compilation
- Stack Protection
- Source Fortification
- ASLR
- Rust, GoLang

Application Isolation
- Cgroups
- Namespaces
- Linux Containers (LXC)
- Docker etc.

Secure Updates
- OSTree
- RAUK
- swupd
- SWUpdate

Network Security
- Firewalls
- TLS
- VPN
- NIDS

Process
- SSDL
- Vulnerability Management
- Maturity Model

Confining Linux Applications with Libseccomp

- All the demo code is available here:

https://github.com/Doulos/EOSS23

Doulos **Secure Embedded** courses use knowledge and skills, gained from the international semiconductor industry, to deliver the right training and support for you and your organization.

Security courses cover:

- Embedded Linux
- Embedded C & C++
- Arm® TrustZone®

- Unique training includes extensive hands-on labs
- Course attendees benefit from a complete learning experience

Find our more: **doulos.com/secureembedded**

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification

» SystemVerilog » UVM » Formal
» SystemC » TLM-2.0

FPGA & Hardware Design

» VHDL » Verilog » SystemVerilog
» Tcl » AMD-Xilinx » Intel FPGA

Embedded Software

» Emb C/C++ » Emb Linux » Yocto
» RTOS » Security » Arm

AI & Deep Learning

SystemVerilog OUT OF THE BOX

UVM

SYSTEMC

yocto PROJECT PARTICIPANT

freeRTOS

SECURE EMBEDDED

arm APPROVED TRAINING PARTNER